INTRODUCING CLASSES, OBJECTS, AND METHODS

Asmaliza Ahzan
IVERSON ASSOCIATES SDN BHD

Introducing Classes, Objects, and Methods

Class Fundamentals

- Class defines a new data type.
- Once defined, this new type can be used to create objects of that type.
- Thus, a class is a template for an object, and an object is an instance of a class.
- A class is declared by use of the class keyword.
- The data, or variables, defined within a class are called instance variables.
- The code is contained within methods.
- Collectively, the methods and variables defined within a class are called members of the class.

```
class classname {
        type instance-variable1;
        type instance-variable2;
        // ...
        type instance-variableN;

        type methodname1(parameter-list) {
                // body of method
        }

        type methodname2(parameter-list) {
                // body of method
        }
        // ...

        type methodnameN(parameter-list) {
                // body of method
        }
}
```

Here is a class called Box that defines three instance variables: width, height, and depth. Currently, Box does not contain any methods.

```
class Box {
      double width;
      double height;
      double depth;
}
```

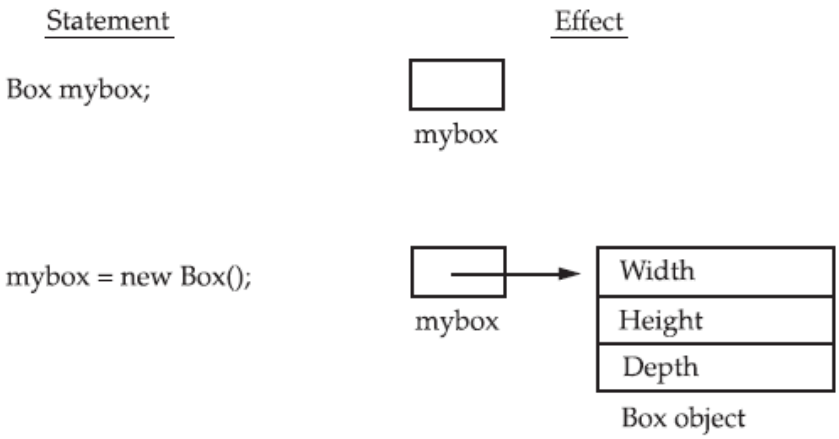
How Objects Are Created

```
// This program declares two Box objects.
class BoxDemo2 {
      public static void main(String args[]) {
            Box mybox1 = new Box();
            Box mybox2 = new Box();
            double vol;

            // assign values to mybox1's instance variables
            mybox1.width = 10;
            mybox1.height = 20;
            mybox1.depth = 15;

            /* assign different values to mybox2's instance variables */
            mybox2.width = 3;
            mybox2.height = 6;
            mybox2.depth = 9;

            // compute volume of first box
            vol = mybox1.width * mybox1.height * mybox1.depth;
            System.out.println("Volume is " + vol);

            // compute volume of second box
            vol = mybox2.width * mybox2.height * mybox2.depth;
            System.out.println("Volume is " + vol);
      }
}
```

Reference Variables and Assignment

As just explained, the new operator dynamically allocates memory for an object. It has this general form:

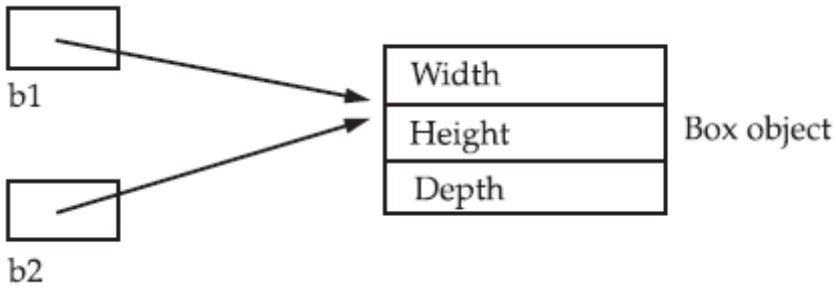
class-var = new *classname* ( );



Object reference variables act differently than you might expect when an assignment takes place. For example, what do you think the following fragment does?

```
Box b1 = new Box();
Box b2 = b1;
```

You might think that b2 is being assigned a reference to a copy of the object referred to by b1. That is, you might think that b1 and b2 refer to separate and distinct objects. However, this would be wrong. Instead, after this fragment executes, b1 and b2 will both refer to the same object. The assignment of b1 to b2 did not allocate any memory or copy any part of the original object. It simply makes b2 refer to the same object as does b1. Thus, any changes made to the object through b2 will affect the object to which b1 is referring, since they are the same object.



Methods

This is the general form of a method:

type name(*parameter-list*) {
    // body of method
}

- Type specifies the type of data returned by the method. This can be any valid type, including class types that you create. If the method does not return a value, its return type must be void.
- The name of the method is specified by name. This can be any legal identifier other than those already used by other items within the current scope.
- The parameter-list is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

```
// This program includes a method inside the box class.
class Box {
      double width;
      double height;
      double depth;

      // display volume of a box
      void volume() {
            System.out.print("Volume is ");
            System.out.println(width * height * depth);
      }
}
```

Returning a value

```
// Now, volume() returns the volume of a box.
class Box {
      double width;
      double height;
      double depth;

      // compute and return volume
      double volume() {
            return width * height * depth;
      }
}
```

Method that takes parameters

```
// This program uses a parameterized method.
class Box {
      double width;
      double height;
      double depth;

      // compute and return volume
      double volume() {
            return width * height * depth;
      }

      // sets dimensions of box
      void setDim(double w, double h, double d) {
            width = w;
            height = h;
            depth = d;
      }
}
```

Constructors

- A constructor initializes an object immediately upon creation.
- It has the same name as the class in which it resides and is syntactically like a method.
- The constructor is automatically called when the object is created before the new operator completes.
- Constructors look a little strange because they have no return type, not even void.
- It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.

```
/* Here, Box uses a constructor to initialize the dimensions of a box. */
class Box {
	double width;
	double height;
	double depth;

	// This is the constructor for Box.
	Box() {
		System.out.println("Constructing Box");
		width = 10;
		height = 10;
		depth = 10;
	}

	// compute and return volume
	double volume() {
		return width * height * depth;
	}
}
```

Parameterized Constructors

```
/* Here, Box uses a parameterized constructor to initialize the dimensions
of a box. */
class Box {
	double width;
	double height;
	double depth;

	// This is the constructor for Box.
	Box(double w, double h, double d) {
		width = w;
		height = h;
		depth = d;
	}

	// compute and return volume
	double volume() {
		return width * height * depth;
	}
}
```

The this keyword

- Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the this keyword.
- this can be used inside any method to refer to the current object.
- That is, this is always a reference to the object on which the method was invoked.

```
// Use this to resolve name-space collisions.
Box(double width, double height, double depth) {
      this.width = width;
      this.height = height;
      this.depth = depth;
}
```

Garbage Collection

- In some languages, such as C++, dynamically allocated objects must be manually released by use of a delete operator.
- Java takes a different approach; it handles deallocation for you automatically.
- The technique that accomplishes this is called garbage collection.
- It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.